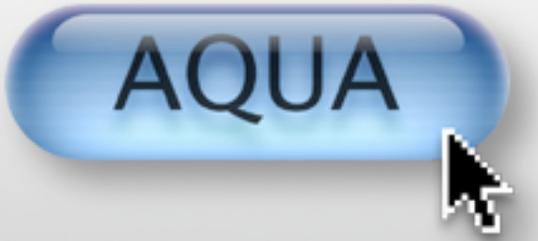
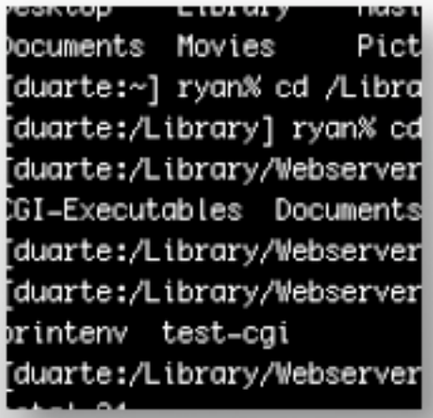
Handling Audio Formats with CoreAudio

Session 405





Handling Audio Formats with CoreAudio

James McCartney
Core Audio Engineering
jmccartney@apple.com

Demo

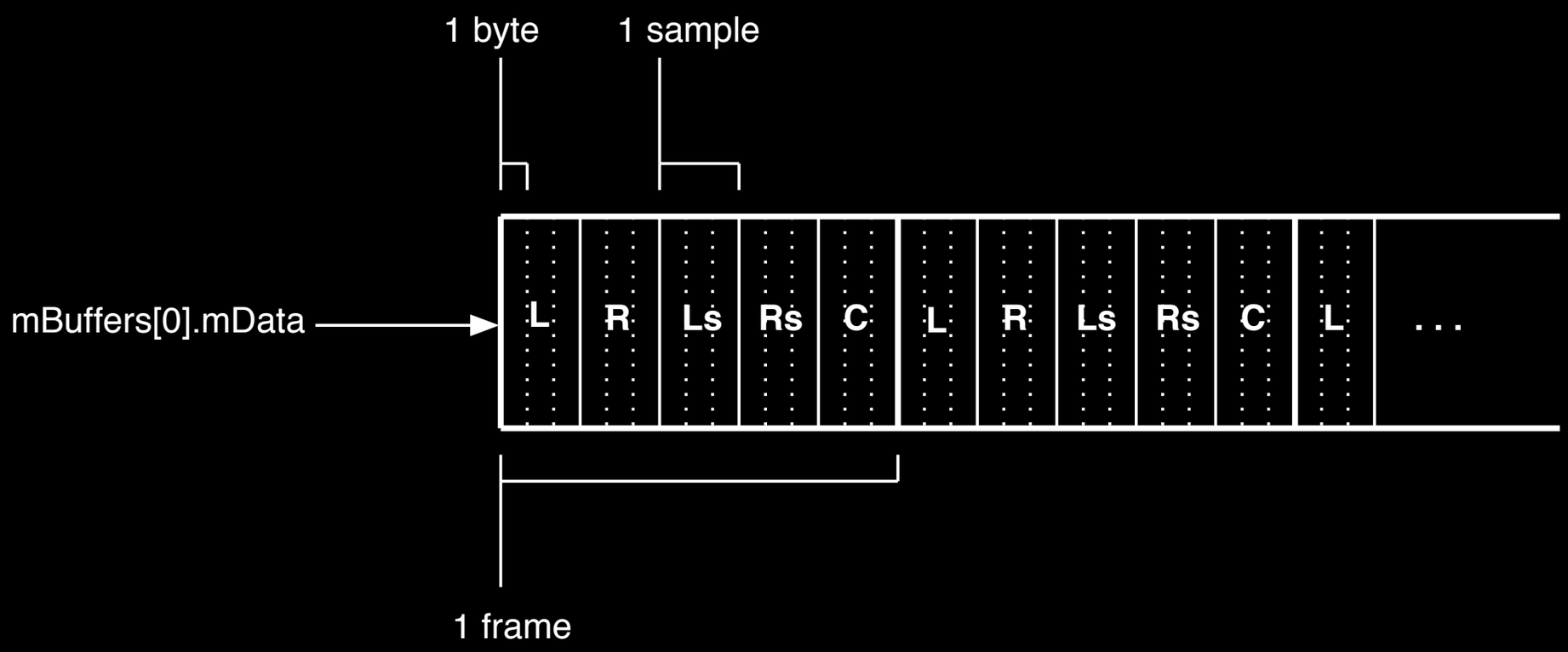
Doug Wyatt

Audio Format Handling APIs

- How formats are represented in CoreAudio
- AudioConverter
- AudioFormat
- Audio Units
  - Multichannel/surround
- AudioFileGetGlobalInfo
- MatrixMixer Audio Unit

Bytes, Samples, Frames, Packets

- For linear PCM, 1 Frame = 1 Packet



AudioStreamBasicDescription

```
struct AudioStreamBasicDescription
{
  Float64 mSampleRate;       //the native sample rate of the audio stream
  UInt32 mFormatID;          //the specific format type of audio stream
  UInt32 mFormatFlags;       //flags specific to each format
  UInt32 mBytesPerPacket;    //the number of bytes in a packet
  UInt32 mFramesPerPacket;   //the number of frames in each packet
  UInt32 mBytesPerFrame;     //the number of bytes in a frame
  UInt32 mChannelsPerFrame;  //the number of channels in each frame
  UInt32 mBitsPerChannel;    //the number of bits in each channel
  UInt32 mReserved;          //reserved padding
};
typedef struct AudioStreamBasicDescriptionAudioStreamBasicDescription;
```

AudioStreamBasicDescription

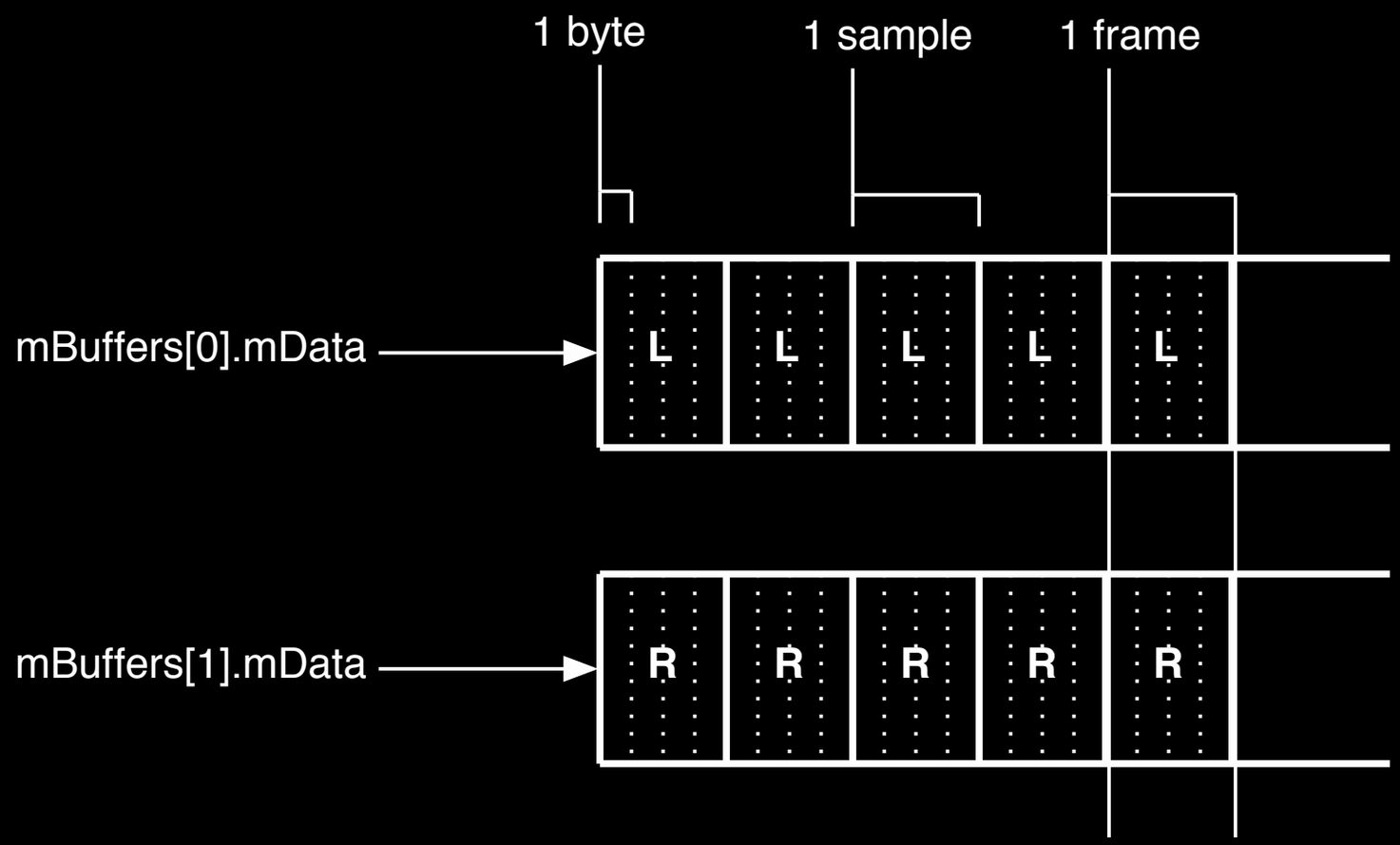
Example: 5-channel 24-bit big-endian integer

```
AudioStreamBasicDescription asbd;

asbd.mSampleRate = 44100.;
asbd.mFormatID = kAudioFormatLinearPCM;
asbd.mFormatFlags = kLinearPCMFormatFlagIsBigEndian
        | kLinearPCMFormatFlagIsSignedInteger
        | kLinearPCMFormatFlagIsPacked;
asbd.mFramesPerPacket = 1;
asbd.mBytesPerPacket = 15;
asbd.mBytesPerFrame = 15;
asbd.mChannelsPerFrame = 5;
asbd.mBitsPerChannel = 24;
```

Bytes, Samples, Frames, Packets

32 bit floating point, stereo, non interleaved



AudioStreamBasicDescription

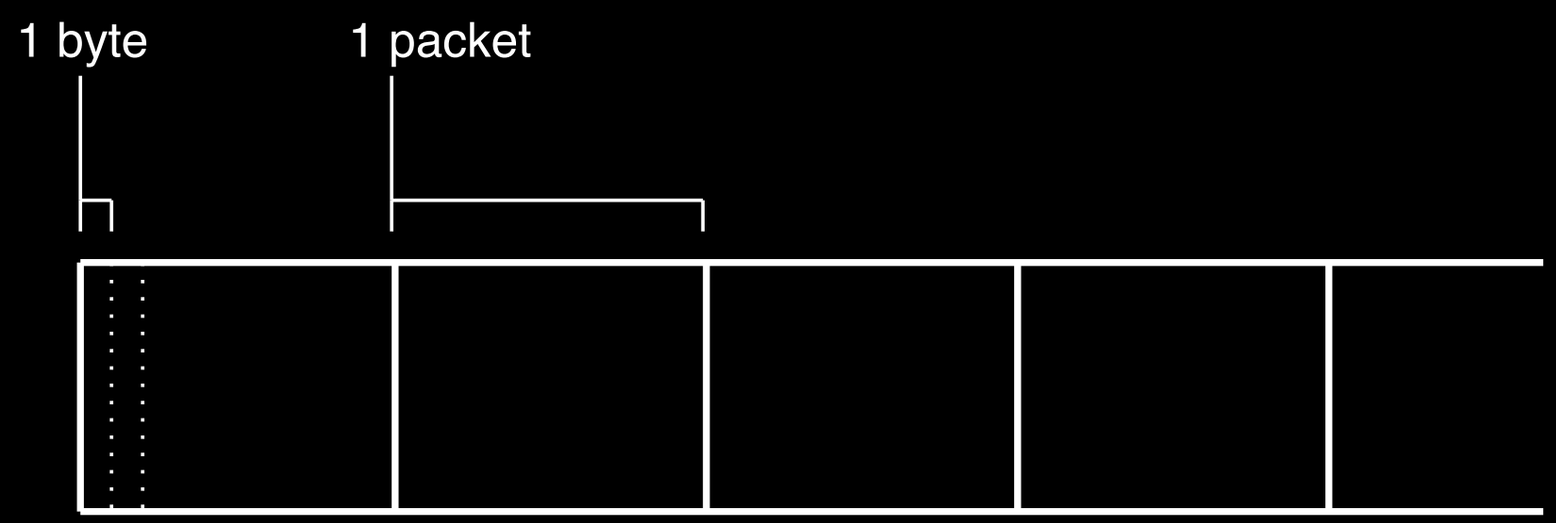
Example: Stereo 32-bit floating-point, noninterleaved

```
AudioStreamBasicDescription asbd;

asbd.mSampleRate = 44100.;
asbd.mFormatID = kAudioFormatLinearPCM;
asbd.mFormatFlags = kLinearPCMFormatFlagIsBigEndian
          | kLinearPCMFormatFlagIsSignedInteger
          | kLinearPCMFormatFlagIsPacked
          | kLinearPCMFormatFlagIsNonInterleaved;
asbd.mFramesPerPacket = 1;
asbd.mBytesPerPacket = 4;
asbd.mBytesPerFrame = 4;
asbd.mChannelsPerFrame = 2;
asbd.mBitsPerChannel = 32;
```

Bytes, Samples, Frames, Packets

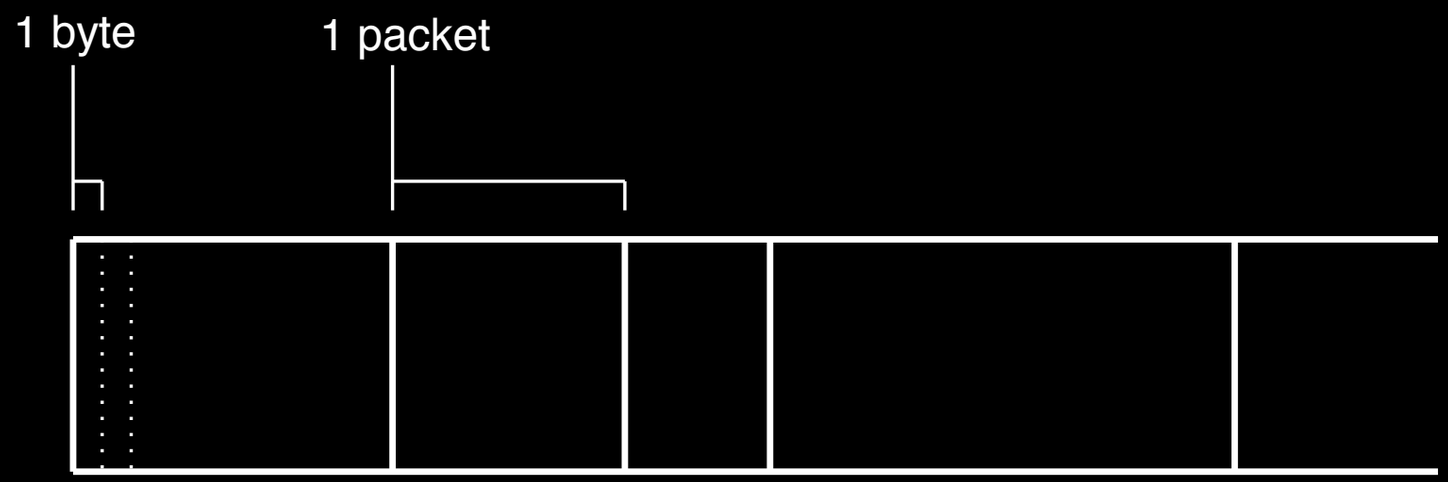
Compressed Constant Bit Rate (CBR) data



mBytesPerPacket is constant

Bytes, Samples, Frames, Packets

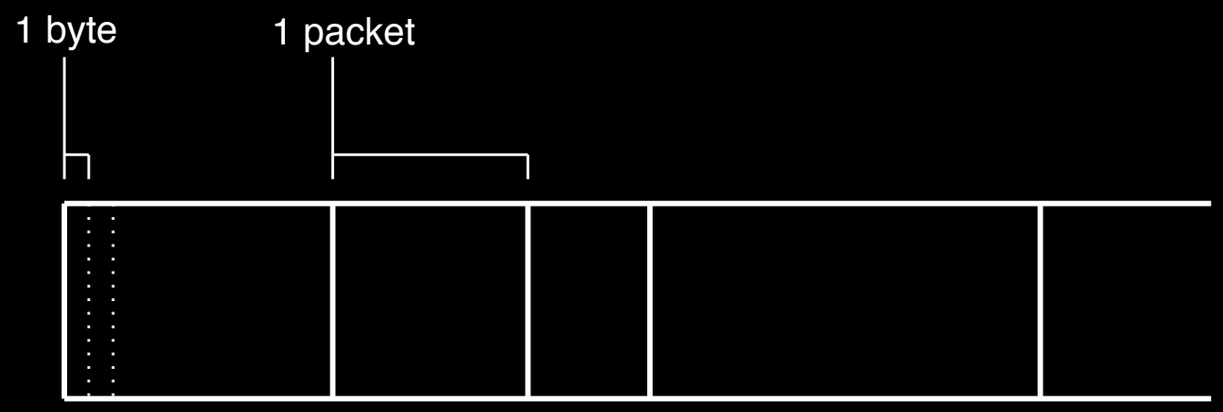
Compressed Variable Bit Rate (VBR) data



mBytesPerPacket is not constant (set to zero)

AudioStreamPacketDescription

- Used to describe packet boundaries for externally framed VBR formats such as AAC
- Starting byte offset of packet
- Length in bytes of packet



AudioBufferList

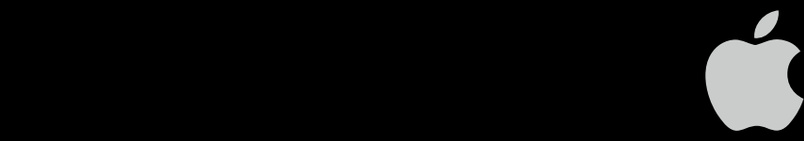
CoreAudio's container for audio data

```
struct AudioBufferList
{
   UInt32    mNumberBuffers;
   AudioBuffer mBuffers[1];  // variable length array
};

struct AudioBuffer
{
   UInt32 mNumberChannels; // number of interleaved channels
   UInt32 mDataByteSize;   // the size of the buffer
   void*  mData;           // the pointer to the buffer
};
```

Where and How

- All of these structs are declared in CoreAudioTypes.h
  - AudioStreamBasicDescription
  - AudioStreamPacketDescription
  - AudioBufferList
  - AudioChannelLayout
- They are used everywhere in CoreAudio
- Make use of CoreAudio SDK Public Utility C++ wrapper classes to make your life simpler



Formats and AudioUnits

- AudioUnits express buffer sizes in frames
- By default the format is 32 bit float, non interleaved
- BUT the API can express other formats for AudioUnit streams

Formats and the HAL

- HAL always counts time in frames
- LPCM mode
  - Buffer size in frames
  - Default is 32 bit float
- Non LPCM mode:
  - 1 packet at a time
  - Buffer frame size range restricted to the number of frames per packet

Formats and AudioFile API

- AudioFile can read and write in bytes or packets
  - AudioFileReadBytes, AudioFileWriteBytes
  - AudioFileReadPackets, AudioFileWritePackets

How Do You Convert from One Format to Another?

AudioConverter

- Converts:
  - Floating-point/integer, various bit depths
  - Sample rate conversion
  - Interleaving/deinterleaving
  - Channel re-ordering
  - PCM <-> compressed formats (new in Panther)

AudioConverter

Creating

AudioConverterNew

```
  AudioStreamBasicDescription in, out;

  .. Fill out stream descriptions (can use SDK class
CAStreamBasicDescription) ..

  AudioConverterRef decoder;
  OSStatus err = AudioConverterNew(&in, &out, &decoder);
```

AudioConverter

Converting buffers

```
OSStatus
AudioConverterFillComplexBuffer(
      AudioConverterRef inAudioConverter,
      AudioConverterComplexInputDataProc inInputDataProc,
      void *inInputDataProcUserData,
      UInt32 *ioOutputDataPacketSize,
      AudioBufferList *outOutputData,
      AudioStreamPacketDescription *outPacketDescription
);
```

AudioConverterFillComplexBuffer

- Preparing the output buffer list
  - Interleaved: one buffer
  - Non-interleaved: multiple mono buffers.
  - Buffers' mData will receive converted audio on output
  - Buffers' mDataByteSize must be large enough to contain the requested number of packets

AudioConverterFillComplexBuffer

```
AudioConverterRef decoder;
AudioBufferList bufList;

... allocate my output buffer ...
... can use SDK class CABufferList to fill out buffer list ..
... enter loop ...

UInt32 ioOutputDataPacketSize = 8192;

err = AudioConverterFillComplexBuffer(decoder, inputProcPtr,
        userData, &ioOutputDataPacketSize, &bufList, NULL);
```

AudioConverterFillComplexBuffer

- Input proc responsibilities
  - Implements the data source of the demand driven model
  - Set up ioData to point to the source data (not copy data!)
  - Don’t overwrite or release the source data until input proc called again
  - May provide more or less than requested. If less, then it will be called

AudioConverterFillComplexBuffer

The input proc

```
OSStatus  MyInputProc(
      AudioConverterRef inAudioConverter,
      UInt32 *ioNumberDataPackets,
      AudioBufferList *ioData,
      AudioStreamPacketDescription
                        **outDataPacketDescription,
      void *inUserData
);
```

AudioConverterFillComplexBuffer

- Input proc—special conditions
  - End-of-stream
    - Set *ioNumberDataPackets to 0. return noErr
    - Proc may still get called again; return 0 packets again
  - No data available right now
    - Return an error, which is propagated back to the caller
    - Any data which was converted will be returned to the caller; converter may keep unconverted data buffered internally

AudioConverterFillComplexBuffer

InputProc example:

```
OSStatus   FromFloatInputProc (
           AudioConverterRef                inAudioConverter,
           UInt32                           *ioNumberDataPackets,
           AudioBufferList                  *ioData,
           AudioStreamPacketDescription     **outDataPacketDescription,
           void                             *inUserData)
{
MyUserData *stuff = static_cast<MyUserData*>(inUserData);
AudioBufferList *bufList = stuff->bufList;
for (UInt32 i=0; i < bufList->mNumberBuffers; ++i)
{
    ioData->mBuffers[i].mNumberChannels = bufList->mBuffers[i].mNumberChannels;
    ioData->mBuffers[i].mData = bufList->mBuffers[i].mData;
    ioData->mBuffers[i].mDataByteSize = bufList->mBuffers[i].mDataByteSize;
}
*ioNumberDataPackets = ioData->mBuffers[0].mDataByteSize
                     / stuff->mInputASBD.mBytesPerPacket;
return noErr;
}
```

Audio Format API

AudioFormat API

- Provides operations for handling:
  - AudioStreamBasicDescription
  - AudioChannelLayout
    - A description of the channels present and their ordering
  - Information about compressed formats
    - Codecs present on the system

AudioStreamBasicDesc. Operations

- Get a format's name
- Fill out a partial stream description
- Is format VBR?
- Is format externally framed?
- What formats can I encode?
- What formats can I decode?

AudioFormat API

Using the AudioFormat API

```
    UInt32 size;
    OSStatus err;
    OSType *formatIDs;

    err = AudioFormatGetPropertyInfo(kAudioFormatProperty_EncodeFormatIDs, 0, NULL,
&size);

    formatIDs = (OSType*)malloc(size);
    UInt32 numFormats = size / sizeof(OSType);

    err = AudioFormatGetProperty(kAudioFormatProperty_EncodeFormatIDs, 0, NULL, &size,
                                 formatIDs);
    if (err) return err;

    for (UInt32 i=0; i<numFormats; ++i)
    {
        AudioStreamBasicDescription absd;
        memset(&absd, 0, sizeof(absd));
        absd.mFormatID = formatIDs[i];

        CFStringRef name;
        size = sizeof(CFStringRef);
        err = AudioFormatGetProperty(kAudioFormatProperty_FormatName, sizeof(absd),
                        &absd, &size, &name);
        if (err) return err;

        CFShow(name);
    }
```

Using the AudioFormat API

Results:

AudioChannelLayout

- Describes the channel ordering of a stream
  - Predefined layouts: integer tag
  - USB/WAVE layouts: bitmap
  - Arbitrary layouts: array of channel descriptions

AudioChannelLayout Struct

```
typedef UInt32 AudioChannelLayoutTag;

struct AudioChannelLayout
{
  AudioChannelLayoutTag    mChannelLayoutTag;
  UInt32                   mChannelBitmap;
  UInt32                   mNumberChannelDescriptions;
  AudioChannelDescription  mChannelDescriptions[1];
};
typedef struct AudioChannelLayout AudioChannelLayout;
```

AudioChannelLayoutTag

- A lot of formats are predefined: mono, stereo, 3.0, 4.0, 5.0, 6.0, 7.0, 3.1, 4.1, 5.1, 6.1, 7.1, mid-side, X-Y, binaural, ambisonic B-format, quad, pentagonal, hexagonal, octagonal, cube
- Low 16 bits gives the number of

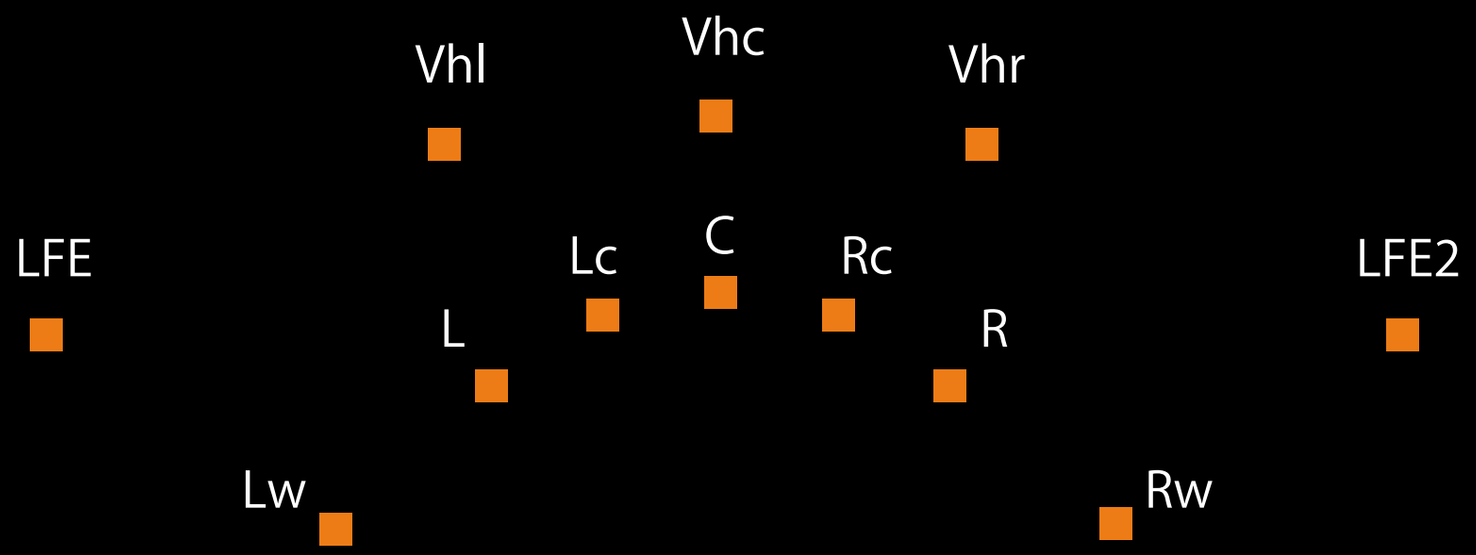
```
enum
{
  kAudioChannelLayoutTag_UseChannelDescriptions  = (0 << 16) | 0,
  kAudioChannelLayoutTag_UseChannelBitmap        = (1 << 16) | 0,
  kAudioChannelLayoutTag_Mono                    = (100 << 16) | 1,
  kAudioChannelLayoutTag_Stereo                  = (101 << 16) | 2,
...
  kAudioChannelLayoutTag_MPEG_5_1_A = (121 << 16) | 6,   // L R C LFE Ls Rs
  kAudioChannelLayoutTag_MPEG_5_1_B = (122 << 16) | 6,   // L R Ls Rs C LFE
  kAudioChannelLayoutTag_MPEG_5_1_C = (123 << 16) | 6,   // L C R Ls Rs LFE
  kAudioChannelLayoutTag_MPEG_5_1_D = (124 << 16) | 6,   // C L R Ls Rs LFE
... about 40 defined layout tags ...
```

AudioChannelDescription Struct

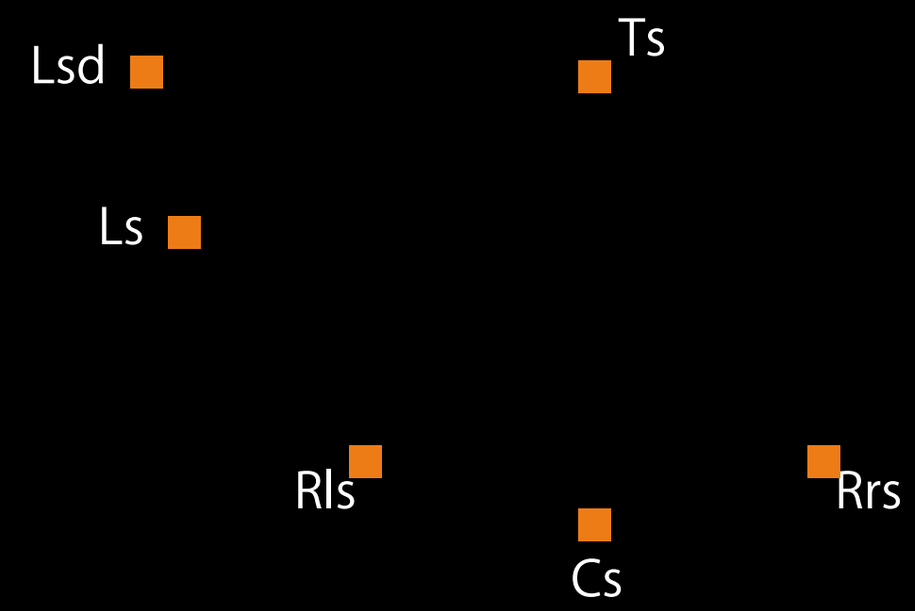
AudioChannelLabel

```
enum
{
  kAudioChannelLabel_Unknown          = 0xFFFFFFFF,
  kAudioChannelLabel_UseCoordinates   = 100,
  kAudioChannelLabel_Unused           = 0,
  kAudioChannelLabel_Left             = 1,
  kAudioChannelLabel_Right            = 2,
  kAudioChannelLabel_Center           = 3,
  kAudioChannelLabel_LFEScreen        = 4,
  kAudioChannelLabel_LeftSurround     = 5,
  kAudioChannelLabel_RightSurround    = 6,
... more than 40 defined channel labels ...
```

AudioChannelLabels



“Left surround direct”



A rough view of some of the defined channel labels

AudioChannelLayout Operations

- Get the number of channels in a layout
- Get a full description from a layout tag or bitmap
- Get a matrix for downmixing
- Get a name for a layout
- Get a name for a channel
- Put an AudioChannelLayout in an AIFF file

AudioChannelLayout in AudioUnits

- Get the channel layouts an AudioUnit supports
  - `kAudioUnitProperty_SupportedChannelLayoutTags`
- Get/set a channel layout for a AU stream
  - `kAudioUnitProperty_AudioChannelLayout`
- MatrixReverb AU supports:
  - Input:
    - 1 or 2 channels
  - Output:
    - Stereo, quad, 5.0

AudioChannelLayout in AudioUnits

- Don’t need for:
  - AU only doing mono or stereo
  - AU doesn’t care about spatial location
- Do need for:
  - Any AU that deals with surround or spatially oriented rendering, to distinguish between:
  - Same numChannels different spatial characteristics (for example, ambisonics-wxyz vs quad)
  - Same spatial layout (eg. 5.0) but different speaker ordering (for instance, L,R,C,Ls,Rs vs L,R,Ls,Rs,C)

ChannelLayouts and AudioConverter

- Allows encoding and decoding surround AAC
  - Apple's AAC codec supports mono, stereo, 4.0, 5.0, 6.0, 7.0, 5.1, 6.1, 7.1
- Get available encode channel layouts
  - `kAudioConverterAvailableEncodeChannelLayouts`
- Get/set codec channel layout
  - `kAudioConverterCodecChannelLayout`

AudioFile Global Info API

- File types that can be read
- File types that can be written
- A name for a file type
- Stream formats that can be read
- All file extensions that can be read
- File extensions for a file type

AudioFile Global Info

```
extern OSStatus
AudioFileGetGlobalInfoSize (
                                AudioFilePropertyID  inPropertyID,
                                UInt32               inSpecifierSize,
                                void                 *inSpecifier,
                                UInt32               *outDataSize
                            );


extern OSStatus
AudioFileGetGlobalInfo(
                                AudioFilePropertyID   inPropertyID,
                                UInt32                inSpecifierSize,
                                void                  *inSpecifier,
                                UInt32                *ioDataSize,
                                void                  *outPropertyData
                            );
```

Using AudioFileGetGlobalInfo

```
OSStatus err;
UInt32 propertySize;
err = AudioFileGetGlobalInfoSize(kAudioFileGlobalInfo_WritableTypes, 0,
                                 NULL, &propertySize);
if (err) return err;

OSType *types = (OSType*)malloc(propertySize);
err = AudioFileGetGlobalInfo(kAudioFileGlobalInfo_WritableTypes, 0, NULL,
                                 &propertySize, types);
if (err) return err;

UInt32 numTypes = propertySize / sizeof(OSType);
for (UInt32 i=0; i<numTypes; ++i)
{
  CFStringRef name;
  UInt32 outSize = sizeof(name);
  err = AudioFileGetGlobalInfo(kAudioFileGlobalInfo_FileTypeName,
                               sizeof(OSType), types+i, &outSize, &name);
  if (err) return err;

  CFShow(name);
}
```

Using AudioFileGetGlobalInfo

Results:

```
AIFC
AIFF
MPEG Layer 3
NeXT/Sun
Sound Designer II
WAVE
AC3
AAC ADTS
```

Matrix Mixer

Matrix Mixer AudioUnit

- N inputs to M outputs
- CPU usage depends on number of nonzero crosspoints, not the size of the matrix
- Metering on inputs, crosspoints, and outputs

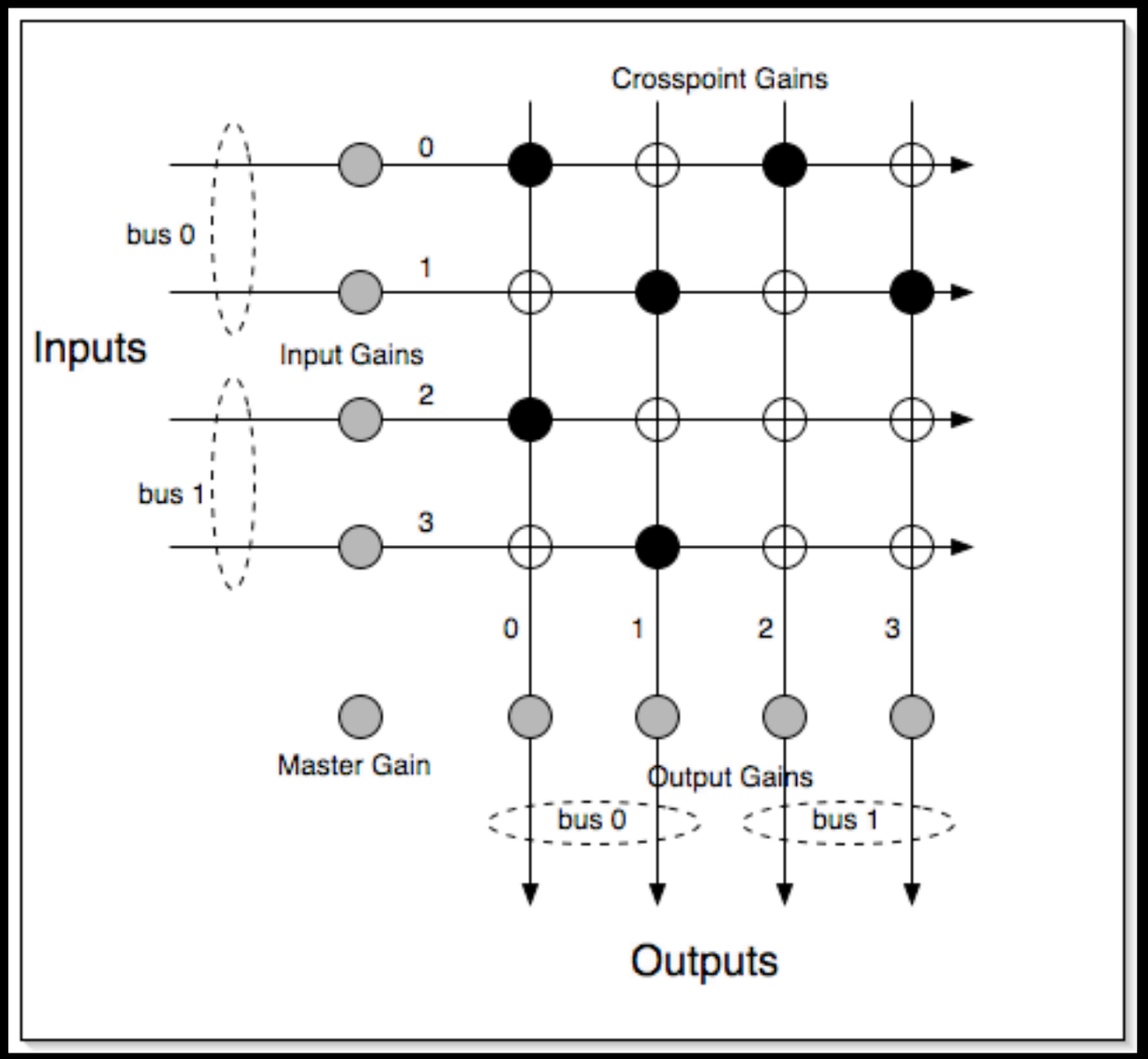
MatrixMixer Applications

- Signal routing
- Channel reordering
- Surround downmixing
- Generalized panning
- And...mixing

Matrix Mixer AudioUnit

- Each channel of an input or output bus is flattened to a set of mono channels. Each crosspoint is a mono to mono connection.
- 4 gains control each crosspoint
  - Input channel gain
  - Crosspoint gain
  - Output channel gain
  - Master gain

Matrix Mixer AudioUnit



Demo

Matrix Mixer AudioUnit Parameters

- All gains can be set from global scope
- Element number specifies which gain
  - Crosspoint: `(inputChannel << 16) | outputChannel`
  - Input gain: `(inputChannel << 16) | 0xFFFF`
  - Output gain: `(0xFFFF << 16) | outputChannel`
  - Master gain: `0xFFFFFFFF`

Matrix Mixer Metering

- Metering is only done if `kAudioUnitProperty_MeteringMode` is set to one
- Inputs: pre and post fader average and peak levels
- Crosspoints and Outputs: post fader avg and peak levels
- Parameters are accessed by element in the same manner as gains

Matrix Mixer Bus Enable

- `kMatrixMixerParam_Enable`
- If a bus is disabled, it will not be pulled
- Disabling a bus smoothly fades the gain to zero before pulling is stopped
- Useful for conserving CPU by shutting down part of the graph
  - Note that setting the input gain to zero will continue pulling the input

Setting Up a Matrix Mixer

- Before initializing, define the dimensions of the matrix
  - Set number of input and output buses
    - kAudioUnitProperty_BusCount
  - Set number of channels in stream formats of each bus
    - kAudioUnitProperty_StreamFormat

AudioUnit: Panner Unit

- AudioUnit Type specifically designed for panning
  - Mono, Stereo, N-Channel Input
  - N-Channel Output
    - Audio Channel Layout
- 2 Modes
  - Panning
  - Channel Volumes
- Developed in conjunction with Mark of the Unicorn
  - http://www.motu.com

Wrap Up

Roadmap

100 IOKit: Kernel Extension Programming Techniques Designing KEXTs for optimal performance	**Russian Hill** **Mon, 3:30pm**
407 Audio Overview	**Russian Hill** **Tues, 3:30pm**
711 Audio and QuickTime	**Russian Hill** **Wed, 9:00am**
FF003 Audio	**North Beach** **Wed, 10:30am**

For More Information

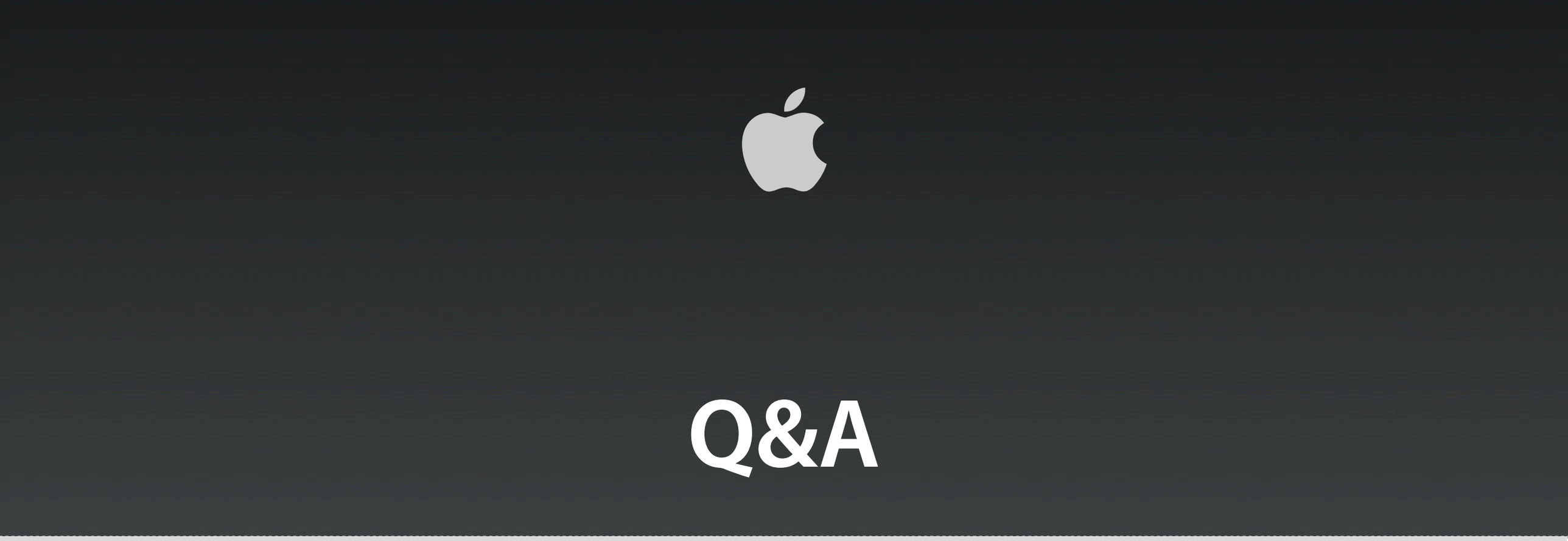
Reference Library

Documentation

Core Audio Documentation
http://developer.apple.com/audio/

Developer Assistance

Core Audio Mailing List
coreaudio-api@lists.apple.com



Q&A

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference